PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F

A2

(11) International Publication Number: **WO 98/43150**

(43) International Publication Date: 1 October 1998 (01.10.98)

(21) International Application Number: PCT/US98/05990

(22) International Filing Date: 26 March 1998 (26.03.98)

(30) Priority Data:
60/042,063 26 March 1997 (26.03.97) US

(71) Applicant *(for all designated States except US)*: DAVOX CORPORATION [US/US]; 9 Technology Park Drive, Westford, MA 01886 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: STRANDBERG, Malcom, B. [–/US]; 65 Blanchard Road, Cambridge, MA 02138 (US). STENT, Robert, J. [–/US]; 28 Salem Road, Westford, MA 01886 (US). CURRERI, Anthony [–/US]; 53 Pilgrim Drive, Litchfield, NH 03051 (US). GILLIS, W., James, Jr. [–/US]; 79 Tewksbury Street, Andover, MA 01810 (US). CAMBRAY, John [–/US]; 7 Christopher Lane, Pelham, NH 03076 (US). SMITH, B., Scott [–/US]; 35 White Plains Avenue, Londonderry, NH 03053 (US).

(74) Agents: BOURQUE, Daniel, J. et al.; Law Offices of Daniel J. Bourque, P.A., Suite 303, 835 Hanover Street, Manchester, NH 03104 (US).

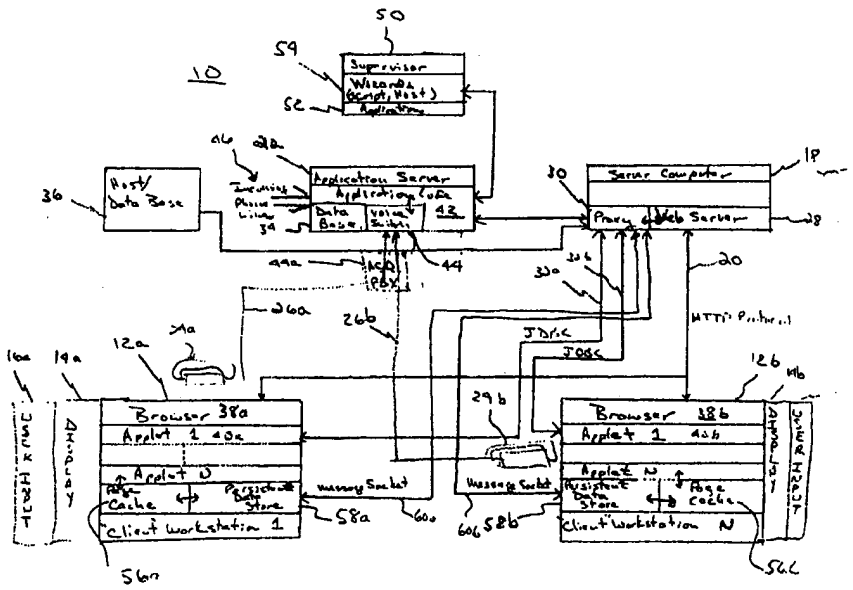
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: BROWSER USER INTERFACE FOR CLIENT WORKSTATION

(57) Abstract

A web browser based client workstation receives web browser based information and displays the web browser based information in a web page browser format. The web browser based information may include operation critical application information which is interactively displayed and utilized to perform an operation critical application from the client workstation. The operation critical application may include a telephony application. The agent workstation may include a script wizard for allowing a user to generate non–proprietary, industry standard, active objects to handle one or more applications. The workstation may also include at least one host connection active object to handle information exchange between the web browser based client workstation and a host computer. Persistent data to be utilized by the client workstation may be stored within the workstation for later reuse without access to a server computer. The client workstation may execute HTML code which may be generated on the client workstation for control of an application by the workstation. The client workstation may also include web browser based scripting information for providing one or more script pages which can be logically related and called one after the other based upon an answer or response to a predetermined message text on a previous script page. The client workstation may be coupled to a server computer wich includes at least one proxy server. The proxy server facilitates access to data external to the server computer. Data from multiple sources can be requested through the proxy server and displayed generally simultaneously on one web page of the web browser based client workstation.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

BROWSER USER INTERFACE FOR CLIENT WORKSTATION

RELATED APPLICATION

This application is related to and claims the benefit of Provisional U.S. Patent Application Number 60/042,063 entitled Browser User Interface for Scripting, filed by the Assignee of the present invention, and incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to computer systems and more particularly, to a browser user interface for a client workstation in a client/server environment in which the client workstation utilizes a web browser type interface for mission critical applications and takes advantage of other web browser interface functions and features.

BACKGROUND OF THE INVENTION

In the past, user interfaces to various computer applications have utilized custom designed, application specific, graphical and non-graphical user interfaces. The problems with these custom design graphical or non-graphical user interfaces are that they are not industry standard, often have a significant training or learning time, and non-uniform product behavior.

Graphical or non-graphical user interfaces must include specific binary code for each computer or operating system type which will utilize the interface. Although such interface code may reside on either a personal computer (PC) or on a file server, this separate network protocol is necessary for distribution of the operating interface. In addition, each computer running the interface will generate its own separate temporary files.

Given the significant volume of software now running on most computers, an additional software package which has its own "look and feel" will require training of the users to

become familiar with the software package.

Accordingly, it would be advantageous to provide an industry standard, uniformly looking, uniformly behaving and uniformly accepted user interface for a mission critical application such as a telephony or other application.

SUMMARY OF THE INVENTION

The present invention features a system including a web browser based operation critical client workstation. The system includes a server computer including at least one web server, for servicing a number of connected web browser based client workstations. At least one client workstation is connected to this server computer and responsive to the web server, receives web browser based information and displays web browser based information in a web page browser format. The web based information received by the client workstation includes operation critical application information which is interactively displayed and utilized to perform an operation critical application directly from the client workstation.

In one embodiment, the operation critical application can include a telephony application such as inbound telephone call servicing, outbound telephone call servicing or a combination of inbound and outbound telephone call servicing applications. The client workstation can include an agent workstation or an agent supervisor workstation. In addition, the operation critical application information may include audio and display animation information.

The invention also features a web browser based operation critical agent workstation for a telephony system which includes a server computer having at least a web server, for servicing a number of connected web browser based client workstations. Also included is at least one telephony system web browser based agent workstation which is coupled to the server computer and responsive to the web

server. The at least one telephony system web browser based agent workstation receives web browser based information and displays the information in a web page browser format. The information received includes operation critical application information which is interactively displayed and utilized to perform an operation critical application such as a telephony application.

In this embodiment, the web browser based agent workstation may include a script wizard, for allowing a user to generate at least one non-proprietary, industry standard active telephony object to handle a telephony application, and a host communication wizard, for allowing the user to generate at least one host connection active object to handle information exchange between the telephony system web browser based agent workstation and a host computer.

In another embodiment, the present invention includes a web browser based operation critical client workstation wherein at least one client workstation is coupled to a web server of a server computer. In response to the web server, at least one client workstation receives web browser based information and displays the information on a web page browser format. The information received by the client workstation includes operation critical application information allowing a user at the client workstation to operate and control the operation critical application from the client workstation. In this embodiment, the client workstation may store persistent data to be used by the workstation within the workstation, for later reuse without accessing the server computer. In yet another embodiment of the present invention, the invention features a web browser based client workstation, coupled to a web server, for executing HTML code and for displaying information in a web page browser format. The client workstation further includes a HTML code generator, for generating on the client workstation, web browser based HTML code for display and

operation on the client workstation. In this embodiment, the HTML code generator may further include a script wizard, for allowing a user to generate at least one non-proprietary, industry standard active object to control an application. This embodiment may further include a host communication wizard, for allowing a user to generate at least one host connection, active object, to handle information exchange between the web browser based client workstation and a host computer.

Another embodiment of the present invention features a telephony system including a web browser based operation critical agent workstation and web browser based scripting. The system includes a server computer having at least a web server, for servicing a number of connected web browser based agent workstations. At least one agent workstation is coupled to the server computer and responsive to the web server for receiving web browser based information and for displaying the information in a web page browser format. The information received by the telephony system web browser based agent workstation includes operation critical telephony application information which is interactively displayed and utilized to perform a telephony operation critical application.

In this embodiment, the web browser based information received by the agent workstation includes web browser based scripting information, for providing at least one script page to be displayed on the agent workstation, for providing information to be communicated to a telephony system customer, and for allowing an agent at the agent workstation to input information regarding the telephony system customer in at least one location in the script page displayed. A further feature of this embodiment may include a number of script pages at least two of which can be displayed in a logical branch structure. In this embodiment, at least a first one of the script pages includes a predetermined

message text comprising at least one question having at least two possible responses. The agent workstation causes a branch to at least a second script page based on an agent's indication of a response to the question in the predetermined message text of the first script page.

In yet another embodiment, the present invention features a telephony system including a web browser based operation critical agent workstation coupled to a computer server including at least a web server. The telephony system web browser based agent workstation receives web browser based information and displays the information in a web page browser format on the workstation. The web page information received by the agent workstation includes operation critical application information which is interactively displayed and utilized to perform a telephony operation critical application.

This embodiment also includes an agent workstation with a script wizard, for allowing a user to generate at least one non-proprietary, industry standard, telephony active object, as well as a host communication wizard, for allowing the user to generate at least one host connection active object for handling information exchange between the telephony system web browser based agent workstation and a host computer. Further, in this embodiment, the operation critical application information includes display of web browser based information for allowing the agent workstation to control both voice and data relative to at least one telephony call.

The present invention features, in another embodiment, a web browser based operation critical client workstation, coupled to a server computer, for receiving and displaying information received from the server computer in a web page browser format. The information received by the web browser based operation critical client workstation includes operation critical application information. In this

embodiment, the server computer includes at least one proxy server, coupled to a source of information external to the server computer, for facilitating access to the data external to the server computer by the client workstation.

In this embodiment, the client workstation includes at least one web browser based web page for communicating with the proxy server on the server computer, for requesting information from the proxy server which is external to the server computer, and for receiving external information from the data source external from the server computer through the proxy server. A further feature of this embodiment allows the web browser based client workstation to access data from multiple sources generally simultaneously, and to display on one web page of the web browser based client workstation the data from multiple sources generally simultaneously.

The present invention additionally includes a system including a web browser based operation critical client workstation wherein the client workstation includes at least one client side active object. The at least one client side active object can initiate and control the operation of at least one function without intervention of a server computer or a web server.

DESCRIPTION OF THE DRAWINGS

These and other features and advantages of the present invention will be better understood by reading the following detailed description, taken together with the drawings wherein:

Fig. 1 is a block diagram of a system including a web browser based client workstation in accordance with the many embodiments of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention features a system 10, Fig. 1, including one or more web browser based client workstations 12. In the preferred embodiment, client workstations 12 include a personal computer having a display 14 and a user input mechanism 16 such as a keyboard, keypad and/or mouse 16. The personal computer (PC) of the client workstation 12 is typically run by an operating system such as Windows based program (Windows 95, Windows 97, Windows NT, etc.) or other operating system such as Unix.

The client workstations 12 are coupled to a server computer 18 by means of an internet type interconnection 20 utilizing a HTTP protocol. In the preferred embodiment, interconnection 20 is an intranet connection and the client workstations 12 are reasonably local to (usually within the same room or building) as server computer 18. This is not, however, a limitation of the present invention as an internet connection 20 may also be utilized thus facilitating the client workstations 12 to be remote from the server computer 18 and the application computer 22, which will be described in greater detail below. In the case where the system 10 of the present invention is utilized to implement a telephony call center as in a disclosed exemplary implementation, the client workstations 12 are generally local to the application server 22 and the server computer 18 given that the agents or other users of the client workstations 12 also require a voice data set 24 such as a telephone for telephone call processing. In the case wherein the client workstations 12 are remote from the application server 22 and/or server computer 18, the client workstations may be coupled to the server computer 18 by means of a first telephone line 20 or by such other means as cable modem and cable line or other similar technology which exists now or may exist in the future. In this embodiment,

a separate telephone line 26 provides voice communication to each of the voice data sets 24 at each of the client workstations 12.

In the exemplary embodiment, server computer 18 is typically a separate kind of stand alone personal computer (PC) which typically services between 10 and 100 client workstations 12. Given the significant processing carried on in the server computer 18, better performance is realized in the system 10 by having a server computer 18 as a separate processor. This is not a tremendous financial burden given the low prices of personal computers at the present time.

However, the present invention contemplates a server computer 18 implemented on or as part of an application server 22, without departing from the scope of the present invention.

Server computer 18 includes a web server 28, as is well known in the art, which provides internet type service using web browser style information written in HTML code as is well known in the art. Server computer 18 also includes proxy server 30 which may interface with web server 28 and provides access by the client workstation 12 to data outside of the server computer 18. One limitation of internet type communication utilizing the HTTP protocol is level of security inherent in such present protocol. This level of security only allows a client workstation to communicate with one server computer 18. Accordingly, a client workstation 12 coupled to a server computer 18 having a particular IP address may only communicate with that web server 28 and may not communicate with and obtain data from outside or external sources.

Accordingly, the present invention features a proxy server 30 which utilizes a JDBC Java code based communication link 32 to each client workstation 12 to access external databases such as database 34 on application

server 22, or a separate host or database 36 remote and external from the web server 28 and server computer 18. In accordance with the present invention, each client workstation 12 includes a browser 38 such as Netscape or Microsoft Explorer. These tasks are considered "lightweight" in terms of processing requirements on the personal computer on which the client workstation 12 is implemented. As such, a high level of performance of the client workstation is essentially guaranteed. Alternatively, instead of a proxy server, a system in accordance with the present invention may include a network file sharer, such as a disk drive, in place of the proxy server, connected between the web server 18 and the application server 22.

In a typical "web page" based information display, each "page" may include one or more applets 40 which are typically Java code based and can be utilized to present "active" objects on the display 14 of each client workstation. Such active objects include buttons which can be pushed, audio, display animation, and other well known active objects.

In accordance with the exemplary embodiment of the present invention, application server 22 is a separate computer such as a personal computer or workstation (such as the type manufactured by Sun Microsystems) which generally includes application software code 42, generally proprietary code written in one or more standard or non-standard programming languages. An application code 42 directs the operation of the system including, in an exemplary embodiment, control of a telephone line switch 44 either internal to the application server 22 or more preferably external to application server 22 in the form of an automated call distributor (ACD) as is well known in the art. In the exemplary embodiment, the voice line switch 44 controls the switching of the line on one of the incoming

telephone lines 46 to one of the voice data sets 24 at each of the client workstations 12. The application server 22 coordinates the switching of voice of a particular telephone call with data associated with that telephone call which may be retrieved by one or more data bases 34, 36 or other data base or host system.

In the present exemplary embodiment, application server 22 also includes an interface to a supervisor workstation 50 which may be another personal computer running one or more applications 52. In the preferred embodiment, supervisor workstation 50 includes one or more "wizards" which is a software programming tool which allows inexperienced users to quickly, easily, and with minimum knowledge of a program language, write various software programs or routines to accomplish various functions.

For example, the present invention features a script wizard which allows a supervisor (for example a telephony agent supervisor) located at supervisor workstation 50 to create scripts and script pages containing scripts for use by the agents. As is well known in the telephony industry, scripts are phrases including both text and active objects which are displayed on the agent's client workstation display 14 to allow the user sitting at the client workstation 12 to perform an operation critical (also sometimes referred to as mission critical) application, such as a telephony application, from the client workstation. An additional wizard 54 provided by the present invention includes a host connection or connectivity wizard which allows the supervisor to easily, quickly and generally effortlessly determine and program first connections for a particular web browser based "page" displayed on the client workstation display 14 to be accessed through proxy server 30 of the server computer 18.

In the present exemplary embodiment of the present invention, all scripts, script pages, close connections, and

the like are stored on the application server 22 and accessed and distributed by the server computer 18 to the client workstations 12.

Utilizing web browser based "internet" features, allows the present invention to present a platform portable, easy to install system and presents to the user of each client workstation 12, a familiar interface to a particular mission critical (considered to be functions critical to the businesses revenue) application. Up until now, using a web browser was limited to casual search, reference and support functions due to the fact that the browser was easy to use and install. However, previous web browser applications were typically not very responsive. Utilizing the present invention, however, and the techniques of the present invention, make the browser display responsive and able to manage the activity and state information required for operation or mission critical business use.

In the prior art, (particularly "mission critical" or essential applications), had their own "look and feel" which were programmed using a graphically user interface and propriety software. This required much learning by the user in order to become familiar with all of the features of the application. The present invention overcomes these limitations by using an internet type browser, such as Netscape or Microsoft Explorer which is now familiar to nearly everyone, and which, in the present invention, is utilized to display web page browser formatted information to the user in a style that he or she is accustomed to seeing. In addition, the users are generally familiar with and in fact need only become familiar with one "set" of tools and toolbars for the browser in order to be able to effectively and efficiently utilize the mission critical application implemented on the system of the present invention. Accordingly, a feature of the present invention is the ability of the present system to provide a web

browser based operation critical client workstation.

Because typical mission critical or operation critical applications typically utilize identical or nearly identical "pages" of web browser based information, the present invention includes a web page cache 56 in each of the client workstations 12. Each web page received from the web server 28 is stored or "cashed" in the cache 56 for later re-use by the browser 38. In this manner, the client workstation 12 does not have to request the page from the web server 28, thus greatly improving system performance.

In connection with page cache 56, another feature of the present invention is the provision of a data store 58 in each of the client workstations. A data store 58 is utilized to store information to be added to or merged with web page information stored in page cache 56. In the present invention, this information is received utilizing a message "socket" 60 in the form of a TRS message from the application server. The following example is useful to explain the data store 58. For example, if one particular mission critical application being run by the client workstation 12 is to handle incoming telephone calls from customers, the application server 22 may be utilized to obtain initial customer identification information from the customer using, for example, DTMF touch tone signals from the customers telephone. Thus, the application server 22 can send a message over message socket 60 to a selected one client workstation 12. That message may include a command to handle an incoming telephone call as well as identification information about the calling customer such as a customer number. This information can be stored in data store 58 and when the client workstation is ready, this information can be merged with a selected web page stored in cache 56 to display, on the client workstation display device 14, specific information about the calling customer.

Utilizing another feature of the present invention

often termed as "push" technology, the data store 58 can provide data to one or more applets 40 running on one or more web pages and page cache 56 to utilize the JDBC connection 32 to proxy server 30 to retrieve more information about the calling party from one or more external data bases 34 or 36. In this manner, although the calling party/customer may have input only his or her customer number, the browser user interface of the present invention can display, on the client workstation display 14, all of the information about the calling party/customer including name, address, telephone number, account balance and history, etc. Alternatively, the application server 22 may utilize information about the calling party's telephone number to lookup information about the calling party without having to request information from the customer. Utilizing such ANI information from an incoming telephone call is well known to those skilled in the art and considered to be within the scope of the present invention. Additionally, application server 22 also coordinates the generally simultaneous arrival of data from one or more external data sources 34, 36 on a display device 14 of a client workstation 12 along with the arrival of voice information over voice data set 24 at the same client workstation 12. Thus, the present invention allows for mission or operation critical applications to be run from at least one client workstation 12.

An example of an application server includes the "Unison Call Center Management System" manufactured by Davox Corporation, the assignee of the present invention, and described, for example, in U.S. Patent Nos. 5,577,112, 5,381,470 and 5,592,543 incorporated herein by reference. In the present description, any reference to Unison is a reference to a Call Center Management System or other application server as is well known in the art. In addition, any reference in the present exemplary description

to "Lyricall" is understood to be a reference to a collection of software programs/tools which implement the exemplary embodiment of the present invention on the Davox Unison brand Call Center Management System, although this not a limitation of the present invention. Other implementations and software, hardware or combinations thereof are considered to be within the scope of the present invention.

Modifications and substitutions by one of ordinary skill in the art are considered to be within the scope of the present invention which is not to be limited except by the claims which follow.

What is claimed is:

Interface davox.admin.AdminInterface

public interface **AdminInterface**

Interface implemented by AdminApplet (and by proxy AdminAppletDemo). The sole purpose of this interface is to allow the hostconn and dataset applets to call into the admin applet by casting an applet to AdminApplet. They must cast to an interface and because of how the demo form of admin is built. In short we end up with two applets AdminApplet which extends Admin AND AdminAppletDemo which extends AdminStub As you can see there is no common base class which could be casted too. Hence the use of this interface. Note we only need in here the methods which will be called from other Java applets. Not those which will be called from JavaScript.

Method Index

Methods

getDBField

```
public abstract String getDBField(String field)
```

setDBField

```
public abstract boolean setDBField(String field,
                                   String value)
```

getTacField

```
public abstract String getTacField(String field)
```

setUserField

```
public abstract void setUserField(String fieldName,
```

String fieldValue)

- **getUserField**

  public abstract String getUserField(String field)

- **getAgentName**

  public abstract String getAgentName()

- **getAgentPassword**

  public abstract String getAgentPassword()

All Packages Class Hierarchy This Package Previous Next Index

Class davox.admin.Admin

```
java.lang.Object
    |
    +----java.awt.Component
            |
            +----java.awt.Container
                    |
                    +----java.awt.Panel
                            |
                            +----java.applet.Applet
                                    |
                                    +----davox.admin.Admin
```

public class **Admin**
extends Applet

Abstract base class to communicate with the Unison WSAPI server. An Admin applet must extend this class and provide implementations for the abstract methods.

Field Index

- **app_name**
- **client_host**
  Agent Logon/Logoff methods.
- **console**
- **form_name**
- **lastStatus**
- **leds**
  LED and debug support
- **LOGIN_STATUS_AVAIL**
- **LOGIN_STATUS_AVAILING**
- **LOGIN_STATUS_BREAK**
- **LOGIN_STATUS_BREAKING**
- **LOGIN_STATUS_LOGGING_OFF**
- **LOGIN_STATUS_LOGGING_ON**
- **LOGIN_STATUS_OFF**
- **LOGIN_STATUS_UNKNOWN**
- **loginStatus**
- **priorStatus**

- **promptCount**
- **props**

Constructor Index

- **Admin**()

Method Index

- **admAvailResponse**(int, String)
- **admCampAddResponse**(int, String)
- **admCampRemoveResponse**(int, String)
- **admCampsResponse**(int, String, String, String)
- **admConnectionLost**()
- **admEnable**(String)
- **admExcludeResponse**(int, String)
- **admLogoffResponse**()
- **admLogonResponse**(int, String)
  Abstract methods which must be defined by a class which extends Admin.
- **admMessage**(String)
- **admPopUp**(String, int)
- **admPrompt**(String, int)
- **admRemoveData**()
- **admSendData**(String)
- **admTacField**(String, String)
- **admUnavailResponse**(int, String)
- **avail**()
- **availResponse**(TrsMsgMsg)
- **campAdd**(String, int)
- **campAddResponse**(TrsMsgMsg)
- **campRemove**(String)
- **campRemoveResponse**(TrsMsgMsg)
- **connectionLost**()
  Handles error when we lose the connection to wsapi.
- **continueLogon**()
  Called from WsapiLogon.class.
- **destroy**()
  Called by browser when applet is being destroyed.
- **exclude**(String, String, String)
  Excludes a record from any further dial attempts
- **excludeResponse**(TrsMsgMsg)
- **getAgentID**()
  Login ID
- **getCampaignName**()

- **getCamps**()
- **getClientHost**()
  Retrieve the ip address of the client system.
- **getDBField**(String)
- **getTacField**(String)
- **getUserField**(String)
- **getVersion**()
  Admin applet version.
- **init**()
  Applet initialization.
- **jsGetCampsResponse**()
- **jsLog**(String)
- **ledError**(int)
- **ledOff**(int)
- **ledOn**(int)
- **logoff**()
  Process a logoff request.
- **logoffResponse**(TrsMsgMsg)
  Called when we receive the message from wsapi accepting or rejecting our request to logoff.
- **logon**(String, String)
  Logon to the Unison system.
- **logPage**(String)
  Logging function for URLs.
- **process**(TrsMsgMsg)
  Process a received message from the WAS.
- **promptRespond**(String)
- **promptRespondTransfer**(String, String)
  Send message with transfer termcode and digits to dial
- **setClientHost**(String)
  Sets the ip address of the client system.
- **setDBField**(String, String)
- **setUserField**(String, String)
- **socketStressTest**()
  Diagnostic method to stress test our JDBC connection.
- **start**()
  Called by browser when applet needs to be started (or restarted).
- **stop**()
  Called by browser when applet needs to be stopped.
- **unavail**()
- **unavailResponse**(TrsMsgMsg)
- **vini**()
  Initialize a connection with WSAPI.
- **viniResponse**()
  Handles the response to the VINI message.

Fields

- **LOGIN_STATUS_UNKNOWN**

  public final static int LOGIN_STATUS_UNKNOWN

- **LOGIN_STATUS_OFF**

  public final static int LOGIN_STATUS_OFF

- **LOGIN_STATUS_BREAK**

  public final static int LOGIN_STATUS_BREAK

- **LOGIN_STATUS_AVAIL**

  public final static int LOGIN_STATUS_AVAIL

- **LOGIN_STATUS_LOGGING_ON**

  public final static int LOGIN_STATUS_LOGGING_ON

- **LOGIN_STATUS_LOGGING_OFF**

  public final static int LOGIN_STATUS_LOGGING_OFF

- **LOGIN_STATUS_BREAKING**

  public final static int LOGIN_STATUS_BREAKING

- **LOGIN_STATUS_AVAILING**

  public final static int LOGIN_STATUS_AVAILING

- **loginStatus**

  public int loginStatus

- **priorStatus**

  public int priorStatus

- **lastStatus**

  public String lastStatus

- **props**

  public Properties props

- **promptCount**

  public int promptCount

- **form_name**

```
public String form_name
```

- **app_name**

```
public String app_name
```

- **console**

```
protected Console console
```

- **client_host**

```
protected String client_host
```

Agent Logon/Logoff methods. Note part of logon include connecting to wsapi and Sybase. When agent logs off we need to close these connections.

- **leds**

```
public AdminLEDs leds
```

LED and debug support

Constructors

- **Admin**

```
public Admin()
```

Methods

- **start**

```
public void start()
```

Called by browser when applet needs to be started (or restarted).

Overrides:
start in class Applet

- **stop**

```
public void stop()
```

Called by browser when applet needs to be stopped. In response we must stop or suspend all running threads.

Overrides:

stop in class Applet

- **destroy**

```
public void destroy()
```

Called by browser when applet is being destroyed. In response we must free any resources which would not otherwise be garbage collected.

Overrides:
destroy in class Applet

- **init**

```
public void init()
```

Applet initialization. Called when applet first created. Performs one time initialization.

Overrides:
init in class Applet

- **process**

```
protected void process(TrsMsgMsg msg)
```

Process a received message from the WAS.

Parameters:
msg - message received from WSAPI

- **setClientHost**

```
public void setClientHost(String client_host)
```

Sets the ip address of the client system. In IE we are unable to determine this from Java code due to security exceptions. To workaround it we use a cgi script to determine the client's ip. One the admin applet is initialized it displays this "setclientip.pl" in the screen frame which will call this method, setClientHost(), then proceed to display the logon prompt in the screen frame.

- **getClientHost**

```
public String getClientHost()
```

Retrieve the ip address of the client system. Call by stats collector.

- **logon**

```
public void logon(String name,
                  String password)
```

Logon to the Unison system. This method will issue a request to the Unison system to logon with

the name *name* and the password *password*. Valid login name/password combinations are attained from the config.dbo.agent_define database table.

This method will change the loginStatus to **LOGIN_STATUS_LOGGING_ON**

Parameters:
name - login name to pass to the Unison system
password - password to pass to the Unison system
Returns:
true if the command was successfully sent, false otherwise.

- **continueLogon**

```
protected void continueLogon()
```

Called from WsapiLogon.class. Continues the login process started when logon() called. See logon() for details.

- **vini**

```
public void vini()
```

Initialize a connection with WSAPI. WSAPI requires a VINI command before a logon command can be sent

- **viniResponse**

```
public void viniResponse()
```

Handles the response to the VINI message. This is where we send the actual agent logon request.

- **logoff**

```
public void logoff()
```

Process a logoff request. This method will invoke a logoff request with a logoff level of LOGOFF_LEVEL_SOFT

- **logoffResponse**

```
public void logoffResponse(TrsMsgMsg msg)
```

Called when we receive the message from wsapi accepting or rejecting our request to logoff.

- **connectionLost**

```
protected void connectionLost()
```

Handles error when we lose the connection to wsapi. Note this is only called from the WsapiRecv thread when an IOException occurs.

- **getVersion**

```
public String getVersion()
```

Admin applet version.

- **getCampaignName**

```
public String getCampaignName()
```

- **getTacField**

```
public String getTacField(String field)
```

- **setUserField**

```
public void setUserField(String fieldName,
                         String fieldValue)
```

- **getUserField**

```
public String getUserField(String field)
```

- **promptRespond**

```
public boolean promptRespond(String text)
```

- **promptRespondTransfer**

```
public boolean promptRespondTransfer(String text,
                                     String digits)
```

Send message with transfer termcode and digits to dial

Parameters:
text - termcode <F08> or <F09>
digits - the digits to dial
Returns:
true

- **getCamps**

```
public void getCamps()
```

- **jsGetCampsResponse**

```
public void jsGetCampsResponse()
```

- **campAdd**

```
public void campAdd(String camp,
                    int level)
```

- **campAddResponse**

```
public void campAddResponse(TrsMsgMsg msg)
```

- **campRemove**

```
public void campRemove(String camp)
```

- **campRemoveResponse**

```
public void campRemoveResponse(TrsMsgMsg msg)
```

- **avail**

```
public void avail()
```

- **availResponse**

```
public void availResponse(TrsMsgMsg msg)
```

- **unavail**

```
public void unavail()
```

- **unavailResponse**

```
public void unavailResponse(TrsMsgMsg msg)
```

- **getDBField**

```
public String getDBField(String field)
```

- **setDBField**

```
public boolean setDBField(String field,
                          String value)
```

- **exclude**

```
public boolean exclude(String app_name,
                       String field_name,
                       String data)
```

Excludes a record from any further dial attempts

Returns:
true

- **excludeResponse**

```
public void excludeResponse(TrsMsgMsg msg)
```

Parameters:

msg - TrsMsgMsg created by exclude

- **jsLog**

```
public void jsLog(String str)
```

- **logPage**

```
public void logPage(String url)
```

Logging function for URLs.

Parameters:
url - the address of the page to be loaded.
Returns:
none

- **getAgentID**

```
public String getAgentID()
```

Login ID

Returns:
agent_id

- **ledOn**

```
public void ledOn(int led)
```

- **ledOff**

```
public void ledOff(int led)
```

- **ledError**

```
public void ledError(int led)
```

- **socketStressTest**

```
public void socketStressTest()
```

Diagnostic method to stress test our JDBC connection. This method loops infinitely sending a bogus termcode for the current call and rereading the database record. In effect, this continously excercises our WSAPI and Sybase connections with both reads and writes.

- **admLogonResponse**

```
protected abstract void admLogonResponse(int statusCode,
                                         String statusString)
```

Abstract methods which must be defined by a class which extends Admin.

- **admSendData**

```
protected abstract void admSendData(String urlString)
```

- **admLogoffResponse**

```
protected abstract void admLogoffResponse()
```

- **admEnable**

```
protected abstract void admEnable(String campaigns)
```

- **admPrompt**

```
protected abstract void admPrompt(String promptString,
                                  int size)
```

- **admPopUp**

```
protected abstract void admPopUp(String promptString,
                                 int size)
```

- **admMessage**

```
protected abstract void admMessage(String promptString)
```

- **admRemoveData**

```
protected abstract void admRemoveData()
```

- **admTacField**

```
protected abstract void admTacField(String tac_field,
                                    String tac_value)
```

- **admCampsResponse**

```
protected abstract void admCampsResponse(int statusCode,
                                         String statusString,
                                         String campsOn,
                                         String campsAvail)
```

- **admCampAddResponse**

```
protected abstract void admCampAddResponse(int statusCode,
                                           String statusString)
```

- **admCampRemoveResponse**

```
protected abstract void admCampRemoveResponse(int statusCode,
                                              String statusString)
```

- **admAvailResponse**

```
protected abstract void admAvailResponse(int statusCode,
                                         String statusString)
```

- **admUnavailResponse**

```
protected abstract void admUnavailResponse(int statusCode,
                                           String statusString)
```

- **admExcludeResponse**

```
protected abstract void admExcludeResponse(int statusCode,
                                           String statusString)
```

- **admConnectionLost**

```
protected abstract void admConnectionLost()
```

All Packages Class Hierarchy This Package Previous Next Index

Overview

The **Admin** applet provides connectivity path to the Unison servers. The **Admin** creates a connection to the Unison servers via the trs.msg package. The **Admin** connects to the Unison WSAPI server. There are subtle differences between the trsmsg connectivity to WSAPI compared to the CLAPI connection. See the differences section for detailed information about the changes.

Functionality

The **Admin** is essentially a Davox specific communication mechanism for browser connectivity to the Unison system. The **Admin** makes use of the trs.msg package to communicate directly to Unison servers. Due to the security restrictions on web browsers the connectivity to Unison servers is limited to the machine which downloads the applet to the browser. This will generate a requirement for a gateway server for connection to Unison servers that reside on more than one physical box.

The **Admin** also provides database connectivity by implementing the jdbc class. The jdbc class (**J**ava **D**atabase **C**onnectivity) allows the **Admin** to communicate directly with the Sybase dataserver. The jdbc

connection is also restricted . .he same connectivity restraints which limit t. .nsg connectivity. Thus, if there is a need for connection to multiple dataservers which reside on separate physical boxes a gateway dataserver will have to be implemented.

The third communication path implemented by the **Admin** is the http communication. This is the native communication mechanism utilized by the web browsers and servers. This is used by the **Admin** to download the custom 'hit' screens.

Methods

Admin

The following list the methods of the Admin class implemented by the **Admin** applet. There are more methods than described below. Those methods are intentionally undocumented as yet their functionality is in a state of flux.

public void process(TrsMsgMsg msg)

This method will process a message buffer received from the Unison server.

public boolean vini()

This method will establish a communication path to the WSAPI server for a browser

public boolean logon(String name, String password)

This method will log the agent into the Unison system with the name and the password.

public void prompt(TrsMsgMsg msg)

This method will process a received message that contains a prompt command.

private void enable(TrsMsgMsg msg)

This method will process the WSAPI enable command. This signifies audio connection -- if applicable has been established with the Unison DSP. This command also contains the list of available campaigns that the agent is allowed to access.

private void sendData(TrsMsgMsg msg)

This method handles a send_data command from the WSAPI server. This functions somewhat differently than the typical WSAPI connection. The **Admin** accepts the command and performs the database lookup itself. The 'form' parameter is the name of an html page to be displayed in the 'data' window of the browser. The html page may have javascript wrapper functions to acquire database information from the **Admin**.

private void removeData(TrsMsgMsg msg)

This method will, indirectly, remove the data from the 'data' window of the browser. This message is usually received in response to a valid INPU command. All database, tac and user fields are removed at this time.

private void tacField sMsgMsg msg)

This method will store tac field messages received by the WSAPI server. The field information can be obtained by the public **getTacField()** method.

public String getTacField(String field)

This method will retrieve the tac field value with the name **field**. If the field is unattainable a null is returned.

public void setUserField(String name, String value)

This method will set a user field. This field will persist until a **removeData()** method is received. The field can be retrieved with the **getUserField()** method.

public void getUserField(String name)

This method will retrieve the value of the user field with name **name**. If the field cannot be obtained a null is returned.

public boolean promptRespond(String text)

This method will issue a prompt response to the WSAPI on behalf of the browser.

public void campAdd(String camp)

This method will issue an add_camp command to the WSAPI server. This will add the user as an agent available to receive calls from the Unison system.

public void campAddResponse(TrsMsgMsg msg)

The response to the add_camp command issued to the WSAPI server.

public void avail()

This method is invoked to take an agent off of break.

public void availResponse(TrsMsgMsg msg)

The response to the avail command.

public void unavail()

This command is issued to put the agent on break.

public void unavailResponse(TrsMsgMsg msg)

This method handles the response to the unavail request.

private void dbReadRecord()

This method will read the Unison call record from the campaign database. The Camp class is used to store the retrieved information.

public boolean setDBField(String field, String value)

This method is used to update any database field in the call record. This method must be called to ensure the data gets written to the database prior to a promptRespond() method invokation.

Database Connectivity

Database connectivity is performed using jdbc. This class allows access to the Sybase dataserver. Since the browsers are still at JVM 1.0.2 this requires obtaining the jdbc class directly from Sybase. The jdbc and the Sybase drivers are required to communicate to the Sybase dataserver. The connectivity for the Admin is performed from the AdminSyb class. The AdminSyb class performs a connection to the Sybase dataserver which houses the campaign database. The AdminSyb also performs the database lookup and update for the **Admin** applet. Database reads are performed for every **sendData()** command and database updates are performed at every **promptRespond()** command.

WSAPI differences

There are only two major points of difference with the WSAPI integration from Admin as opposed to a CLAPI integration. The first difference is in data presentation. The Admin is expecting the form to be an html page. The Admin will be responsible for passing the form data information on to the browser. The second major difference is that the Admin will manage database record integrity. The WSAPI will not be updating the database.

Notes

- The Admin applet requires the Unison system to have a univtac with module was.c at was.c@@/main/uni2.1rel/4 or higher.
- The Admin applet requires a new version of WSAPI that handles trsmsg connections.

Class davox.admin.AdminApplet

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Panel
                           |
                           +----java.applet.Applet
                                   |
                                   +----davox.admin.Admin
                                           |
                                           +----davox.admin.AdminApplet
```

public class **AdminApplet**
extends Admin
implements ActionListener, AdminInterface

Java implementation of the admin bui panel. Presents agent controls and handles agent input events.

Constructor Index

- **AdminApplet**()

Method Index

- **actionPerformed**(Object)
  Callback when one of the buttons in the admin applet is pressed.
- **addCampCookie**(String)
  Called when camps.html is first displayed.
- **admAbortPrompt**()
  Abort a prompt and return agent back to call screen.
- **admAbortTransfer**()
  Callback when cancel selected in confirm transfer prompt
- **admAddCamp**(String)
  Add a campaign to the agent's portfolio.
- **admAvail**()
  Request to make agent available.

- **admAvailResponse**(int, String)
  Callback from Unison to agent "ready to recieve calls" request
- **admCampAddResponse**(int, String)
  Called by admin in response to a campaign being added.
- **admCampRemoveResponse**(int, String)
  Called by admin in response to a campaign being removed.
- **admCampsResponse**(int, String, String, String)
  Called by admin asynchronously after getCamps has been called.
- **admCancelCamps**()
  Called when the "X" is selected in campaign selection page.
- **admCancelPrompt**()
  Callback when X selected in extension.html.
- **admConferenceTransfer**(String)
  Called from JavaScript when a term code is selected.
- **admConfirmTransfer**()
  Callback when check mark selected in confirm transfer prompt
- **admConnectionLost**()
  Handle abnormal losing of connection to Unison server(s).
- **admEnable**(String)
  Agent is logged on and ready to recieve calls
- **admExcludeResponse**(int, String)
  Called by Admin when an EXCLUSION term code response has been processed.
- **admGotoTab**(String)
  Call via JavaScript from "logandgo.html" to change from the call tab to another tab.
- **admLogoff**()
  Handle agent logoff request.
- **admLogoffResponse**()
  Handle the logoff response from Unison.
- **admLogon**()
  Presents the agent logon screen.
- **admLogonResponse**(int, String)
  Handle the logon response from Unison.
- **admMessage**(String)
  Handle messages (was_send_msg).
- **admPopUp**(String, int)
  Handle popup messages (was_popup).
- **admProcessCamps**()
  Called when the check mark is selected in the campaign page.
- **admProcessExclude**(String, String, String)
  Called when check selected in exclude screen
- **admProcessLogon**(String, String)
  Handle agent logon request.
- **admProcessPrompt**(String)
  Callback when check selected in extension.html.
- **admProcessRecall**(String)
  Callback when check mark selected in recall prompt.
- **admProcessSilentTransfer**(String, String)

Called from JavaScript to perform a tranfer without prompting for a number.

- **admProcessTransfer**(String)
  Callback when check mark selected in transfer prompt.
- **admPrompt**(String, int)
  Handle prompt messages.
- **admPromptRespond**(String)
  Called from JavaScript when a term code is selected.
- **admPromptRespondExclude**()
  Called from JavaScript when an EXCLUDE term code is selected.
- **admPromptRespondRecall**(String)
  Called from JavaScript when a RECALL term code is selected.
- **admPromptRespondTransfer**(String)
  Called from JavaScript when a TRANSFER term code is selected.
- **admRemoveCamp**(String)
  Removes a campaign to the agent's portfolio.
- **admRemoveData**()
  Called by Unison to clear the hit screen and enter the "waiting for call" state.
- **admSendData**(String)
  Called by Unison to display a hit screen.
- **admTacField**(String, String)
  Called by Unison once for every TAC field associated with a new call.
- **admUnavailResponse**(int, String)
  Callback from Unison to agent's break request.
- **busyOff**()
  Turn off the busy indicator
- **busyOn**()
  Turn on the busy indicator
- **clearTacStatus**()
  Clears the tac status frame.
- **getAgentMessages**()
  Return AgentMessages for formatting agent message page.
- **getAgentName**()
  Allows agent name to be retrieved from other Java applets or JavaScript.
- **getAgentPassword**()
  Allows agent password to be retrieved from other Java applets or JavaScript.
- **getAvailCamp**()
  Return the next available campaign.
- **getAvailCampCount**()
  Called from camps.html to begin the process of retrieving available campaigns.
- **getCampCookie**()
  Returns the list of selected campaigns for setting the campaigns cookie.
- **getExclusionData**()
  Return ExclusionData for formatting exclude prompt.
- **getJoinedCamp**()
  Return the next selected campaign.
- **getJoinedCampCount**()
  Called from camps.html to begin the process of retrieving selected campaigns.

- **getTacField**()
  Get a tac field.
- **getTacFieldCount**()
  Called from tac.html to retrieve the number of tac fields.
- **getTacStatus**()
  Called from tacstatus.html as a result of showTacStatus() being called.
- **getTransferMsg**()
  Returns text to be displayed in head of transfer.html.
- **handleEvent**(Event)
  Handle mouse events on the button.
- **init**()
  Initialize the admin applet.
- **paint**(Graphics)
  Paint the background graphic.
- **setStatsAvailable**(boolean)
  Called by the stats collector applet when stats are available to be displayed.
- **showTacStatus**(String)
  Displays the message in the tac status frame.
- **showUrl**(String, String)
  Instructs the browser to show a html file in the specified frame.
- **update**(Graphics)
  Override update to prevent flashing.

Constructors

AdminApplet

```
public AdminApplet()
```

Methods

init

```
public void init()
```

Initialize the admin applet. Creates the interface components and does a super.init() to let Admin.class do it's thing.

Overrides:
init in class Admin

actionPerformed

```
public void actionPerformed(Object obj)
```

Callback when one of the buttons in the admin applet is pressed. Dispatches the event for

processing. The switching of tabs is complicated because when we go from the call tab to any other tab we want the url of the current hit screen such that when the call tab is reselected we return the agent to that url and not the start of the script. To do this must go through JavaScript to get the url of the current hit screen. This is done by displaying the file "logandgo.html" in a non viewable frame. This file calls admGotoTab passing it the url of the current hit screen. Only when admGotoTab is called do we actually change tabs. The url of the current hit screen is saved for use if the call tab is reselected.

Parameters:
obj - The button which was pressed.

- **admGotoTab**

```
public synchronized void admGotoTab(String url)
```

Call via JavaScript from "logandgo.html" to change from the call tab to another tab. The route through JavaScript is used to allow the url of the current hit screen to be detected. Note, the termcode prompts have been also added as we know use logandgo as the route to display termcode prompts. This is such that if a termcode prompt is aborted we can return the agent to the correct url of the hit screen also.

Parameters:
url - Url of current hit screen.

- **setStatsAvailable**

```
public void setStatsAvailable(boolean b)
```

Called by the stats collector applet when stats are available to be displayed. We don't enable the stats tab until then.

- **paint**

```
public void paint(Graphics g)
```

Paint the background graphic. Note the PictureButtons extend Canvas. They recieve their own paint messages.

Overrides:
paint in class Container

- **update**

```
public void update(Graphics g)
```

Override update to prevent flashing.

Overrides:
update in class Component

- **handleEvent**

```
public boolean handleEvent(Event e)
```

Handle mouse events on the button. If state of the button will be set according to the event and the button's current state. The event listener for this button may be called.

Overrides:
handleEvent in class Component

- **admSendData**

```
protected void admSendData(String urlString)
```

Called by Unison to display a hit screen. We force the call tab to be visible.

Parameters:
url - Name of html file to be displayed.
Overrides:
admSendData in class Admin

- **admRemoveData**

```
protected void admRemoveData()
```

Called by Unison to clear the hit screen and enter the "waiting for call" state.

Overrides:
admRemoveData in class Admin

- **admTacField**

```
protected void admTacField(String tac_field,
                           String tac_value)
```

Called by Unison once for every TAC field associated with a new call. Saves them away in tacFields. Note we save both the field and value names in the same vector.

Parameters:
tac_field - Field name.
tav_value - Field value.
Overrides:
admTacField in class Admin

- **getTacFieldCount**

```
public synchronized int getTacFieldCount()
```

Called from tac.html to retrieve the number of tac fields. We return tac fields to JavaScript one at a time because in IE 3.0 we cannot return arrays to JavaScript. Note because we store tac fields in a

single vector this is actually 2 * the number of tac fields.

- **getTacField**

```
public synchronized String getTacField()
```

Get a tac field. Note on alternating calls we return the name of the tac field then it's value.

- **admLogon**

```
public void admLogon()
```

Presents the agent logon screen.

- **admProcessLogon**

```
public void admProcessLogon(String uname,
                            String passwd)
```

Handle agent logon request. Called by "check" button in logon page (logon.html). Note the prompting for agent extension (if any) is triggered by an asynchronous prompt callback message from the admin applet

Parameters:
uname - Username entered by agent
passwd - Password entered by agent

- **admLogonResponse**

```
protected void admLogonResponse(int statusCode,
                                String statusString)
```

Handle the logon response from Unison. The only button we enable is logoff. The other buttons will be enabled when admEnable is called by Unison.

Parameters:
statusCode - ADM_SUCCESS or ADM_FAILURE
statusString - If ADM_FAILURE, the reason for the failure.
Overrides:
admLogonResponse in class Admin

- **admLogoff**

```
protected void admLogoff()
```

Handle agent logoff request. Turns the logoff button's busy indicator on to let agent know a logoff is pending.

- **admLogoffResponse**

```
protected void admLogoffResponse()
```

Handle the logoff response from Unison.

Parameters:
statusCode - ADM_SUCCESS or ADM_FAILURE
statusString - If ADM_FAILURE, the reason for the failure.
Overrides:
admLogoffResponse in class Admin

- **admConnectionLost**

```
protected void admConnectionLost()
```

Handle abnormal losing of connection to Unison server(s). When this occurs we display a "Connection lost..." message (in connectlost.html).

Parameters:
statusCode - Not used.
statusString - Not used.
Overrides:
admConnectionLost in class Admin

- **admEnable**

```
protected void admEnable(String campaigns)
```

Agent is logged on and ready to recieve calls

Overrides:
admEnable in class Admin

- **getAgentName**

```
public String getAgentName()
```

Allows agent name to be retrieved from other Java applets or JavaScript.

- **getAgentPassword**

```
public String getAgentPassword()
```

Allows agent password to be retrieved from other Java applets or JavaScript.

- **admUnavailResponse**

```
protected void admUnavailResponse(int code,
                                  String status)
```

Callback from Unison to agent's break request.